Rails (Anti-)Patterns

How not to design your database

{
    "Brad Urani" : "Staff Engineer",
    "Company" : "PROCORE",
}

TOUGH DECISIONS AHEAD
What is Scale?

20,000,000 rows scanned / sec
30,000 transactions / sec
6 TB
1.5 TB RAM
64 Cores
8 TB HD
### CREATE NEW BUDGET LINE ITEM

<table>
<thead>
<tr>
<th>Sub Job</th>
<th>Cost Code</th>
<th>Category</th>
<th>Original Budget</th>
<th>Revised Budget</th>
<th>Committed Costs</th>
<th>Direct Costs</th>
<th>Pending Cost Changes</th>
<th>Projected Costs</th>
<th>Forecast To Complete</th>
<th>Estimated Cost At Completion</th>
<th>Estimated Over/Under</th>
<th>Projected Over/Under</th>
<th>Job To Date Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>Select a Cost Code</td>
<td></td>
<td>$1,490,000.00</td>
<td>$1,490,000.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$1,490,000.00</td>
<td>$1,490,000.00</td>
<td>$0.00</td>
<td>$1,371,975.41</td>
</tr>
<tr>
<td>01-010004 - A/E Fees</td>
<td>S</td>
<td></td>
<td>$2,100.00</td>
<td>$2,100.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$2,100.00</td>
<td>$0.00</td>
<td>$2,100.00</td>
<td>$2,100.00</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>01-010005 - A/E Reimbursables</td>
<td>S</td>
<td></td>
<td>$200.00</td>
<td>$200.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$200.00</td>
<td>$0.00</td>
<td>$200.00</td>
<td>$200.00</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>01-010006 - A/E Reimbursables</td>
<td>S</td>
<td></td>
<td>$6,000.00</td>
<td>$6,000.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$6,000.00</td>
<td>$0.00</td>
<td>$6,000.00</td>
<td>$6,000.00</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>01-010007 - Administrative Personnel</td>
<td>S</td>
<td></td>
<td>$1,000.00</td>
<td>$1,000.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$1,000.00</td>
<td>$0.00</td>
<td>$1,000.00</td>
<td>$1,000.00</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>01-010008 - Administrative Personnel</td>
<td>S</td>
<td></td>
<td>$3,089,355.00</td>
<td>$3,089,355.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$3,089,355.00</td>
<td>$0.00</td>
<td>$3,089,355.00</td>
<td>$3,089,355.00</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>01-010009 - Administrative Personnel</td>
<td>O</td>
<td></td>
<td>$489,355.00</td>
<td>$489,355.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$489,355.00</td>
<td>$0.00</td>
<td>$489,355.00</td>
<td>$489,355.00</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>01-010010 - Administrative Personnel</td>
<td>PD</td>
<td></td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>01-010006 - Business Administration - Personnel</td>
<td>S</td>
<td></td>
<td>$36,600.00</td>
<td>$36,600.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$36,600.00</td>
<td>$0.00</td>
<td>$36,600.00</td>
<td>$36,600.00</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>01-010006 - Business Administration - Personnel</td>
<td>S</td>
<td></td>
<td>$36,600.00</td>
<td>$36,600.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$36,600.00</td>
<td>$0.00</td>
<td>$36,600.00</td>
<td>$36,600.00</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>01-010006 - Business Administration - Personnel</td>
<td>Z</td>
<td></td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>01-010006 - Business Administration - Personnel</td>
<td>O</td>
<td></td>
<td>$30,000.00</td>
<td>$30,000.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$0.00</td>
<td>$30,000.00</td>
<td>$0.00</td>
<td>$30,000.00</td>
<td>$30,000.00</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
</tbody>
</table>
Don't forget developer happiness
A shark will probably eat me.
Trees
Rails doesn't scale

- That's why they don't use it for self-driving cars
  - I could build a self-driving car in a weekend
  - Something Something blockchain
  - OMG React React React

- React! React! OMG React!
  - "1" == 1

- Just add more servers
  - But remember Twitter
    - You're not Twitter
    - Elixir
  - It does if you generate good SQL
Rails doesn't scale

- That's why they don't use it for self-driving cars
- I could build a self-driving car in a weekend
- Something Something blockchain

React! React! OMG React!

- "1" == 1
- OMG React React React

- But remember Twitter
  - You're not Twitter
  - Elixir

Just add more servers

- It does if you generate good SQL
(Anti)-Pattern

Naive Tree
<table>
<thead>
<tr>
<th>id</th>
<th>manager_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
SELECT *
FROM employees e1
LEFT JOIN employees e2 ON e1.id = e2.manager_id
LEFT JOIN employees e3 ON e2.id = e3.manager_id
LEFT JOIN employees e4 ON e3.id = e4.manager_id
LEFT JOIN employees e5 ON e4.id = e5.manager_id
WHERE e1.manager_id IS NULL;
SELECT * FROM employees e WHERE e.manager_id IS NULL;

SELECT * FROM employees e WHERE e.manager_id IN (1);

SELECT * FROM employees e WHERE e.manager_id IN (2,3,4);

SELECT * FROM employees e WHERE e.manager_id IN (5,6,7,8,9);

SELECT * FROM employees e WHERE e.manager_id IN (10,11,12,13,14)
WITH cte_name (column1, column2, …)
AS
( cte_query_definition -- Anchor member
UNION ALL
cte_query_definition -- Recursive member;
references cte_name. )
-- Statement using the CTE
SELECT *
FROM   cte_name
@BradUrani
Recurse or suffer!!!
WITH RECURSIVE descendants AS
( SELECT parent, child AS descendant, 1 AS level
  FROM employees
  UNION ALL
  SELECT d.parent, s.child, d.level + 1
  FROM descendants AS d JOIN source s ON d.descendant = s.parent
)
SELECT *
FROM descendants
ORDER BY parent, level, descendant;
The Mob

- Tony Soprano
  - Larry Boy
    - Alfie Boy
  - Bobby Bacala
  - Paulie Walnuts
    - Chris Moltisanti
      - Alfie
      - Bissel
      - Murmur
    - Little Jimmie
WITH RECURSIVE CTE AS (  
SELECT *  
  , nextval('erp_cost_codes_id_seq') AS new_id  
  , ARRAY[cc.id] as path  
  , false as cycle  
FROM cost_codes cc  
WHERE cc.biller_id = #{biller_id}  
AND cc.biller_type = '#{biller_type}'  
AND cc.parent_id IS NULL  
AND cc.deleted_at IS NULL  
UNION ALL  
SELECT *  
  , nextval('erp_cost_codes_id_seq') AS new_id  
  , path || cc.id  
  , cc.id = ANY(path)  
FROM CTE  
INNER JOIN cost_codes cc ON CTE.id = cc.parent_id  
WHERE cc.biller_id = #{biller_id}  
AND cc.biller_type = '#{biller_type}'  
AND cc.deleted_at IS NULL  
AND NOT cycle  
)  

INSERT INTO erp_cost_codes (id, parent_id, biller_id, biller_type, procore_cost_code_id, code, name, sortable_code, parent_sortable_code, standard_cost_code_id, created_at, updated_at)  
SELECT C1.new_id as id  
  , C2.new_id as parent_id  
  , #{erp_biller_id || 'NULL'} AS biller_id  
  , #{erp_biller_type ? """" + erp_biller_type + """" : 'NULL'} AS biller_type  
  , C1.as procore_cost_code_id  
  , C1.code  
  , C1.name  
  , C2.sortable_code as parent_sortable_code  
  , C1.standard_cost_code_id  
  , clock_timestamp() as created_at  
  , clock_timestamp() as updated_at  
FROM CTE C1  
LEFT JOIN CTE C2 on  
C1.parent_id = C2.id  
ON CONFLICT DO NOTHING;
WITH RECURSIVE CTE AS (  
    SELECT cc.*,  
        nextval('erp_cost_codes_id_seq') AS new_id  
    , ARRAY[cc.id] as path  
    , False AS cycle  
    FROM cost_codes cc  
    WHERE cc.biller_id = #{biller_id}  
    AND cc.biller_type = '#{biller_type}'  
    AND cc.parent_id IS NULL  
    AND cc.deleted_at IS NULL  
    UNION ALL  
    SELECT cc.*,  
        nextval('erp_cost_codes_id_seq') AS new_id  
    , path || cc.id  
    , cc.id = ANY(path)  
    FROM CTE  
    INNER JOIN cost_codes cc ON CTE.id = cc.parent_id  
    WHERE cc.biller_id = #{biller_id}  
    AND cc.biller_type = '#{biller_type}'  
    AND cc.deleted_at IS NULL  
    AND NOT cycle)  

INSERT INTO erp_cost_codes (id, parent_id, biller_id, biller_type, procost_code_id, code, name, sortable_code, parent_sortable_code, standard_cost_code_id, created_at, updated_at)  
SELECT C1.new_id as id  
    , C2.new_id as parent_id  
    , #{erp_biller_id || 'NULL'} AS biller_id  
    , #{erp_biller_type ? "" + erp_biller_type + "" : 'NULL'} AS biller_type  
    , C1.id as procost_code_id  
    , C1.code  
    , C1.name  
    , C1.sortable_code  
    , C2.sortable_code as parent_sortable_code  
    , C1.standard_cost_code_id  
    , clock_timestamp() as created_at  
    , clock_timestamp() as updated_at  
FROM CTE C1  
LEFT JOIN CTE C2 on  
    C1.parent_id = C2.id  
ON CONFLICT DO NOTHING;
Naive Trees

- Simple
- Referential Integrity
- Disk Space
Naive Trees

- You don't need descendents
- You can cache (changes don't happen often)
Closure Tree
<table>
<thead>
<tr>
<th>id</th>
<th>parent_id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NULL</td>
<td>/</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>music</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>photos</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>rap</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>rock</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>polka</td>
</tr>
<tr>
<td>id</td>
<td>parent_id</td>
<td>name</td>
</tr>
<tr>
<td>----</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>1</td>
<td>NULL</td>
<td>/</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>music</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>photos</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>rap</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>rock</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>polka</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ancestor_id</th>
<th>descendent_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
Descendents

```sql
SELECT f.*
FROM folders f
JOIN folder_trees ft ON f.id = ft.descendant_id
WHERE t.ancestor_id = 2;
```
Performance:

- ancestors: 1 SELECT.
- descendants: 1 SELECT.
- siblings: 1 SELECT.
Easily and efficiently make your ActiveRecord models support hierarchies  

https://closureretree.github.io/closure...

closure-tree  ruby  nested-hashes  tree-structure  rails  activerecord  ancestry  nested-set  hierarchy  hierarchies  descendants

parent  child

870 commits  12 branches  80 releases  51 contributors  MIT

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>gemfiles</td>
<td>prep 6.5.0, add ar 5.1 appraisal</td>
<td>4 months ago</td>
</tr>
<tr>
<td>img</td>
<td>added docs for digraph and updated to 4.1.0.rc1</td>
<td>4 years ago</td>
</tr>
<tr>
<td>lib</td>
<td>prep v6.6.0</td>
<td>2 months ago</td>
</tr>
<tr>
<td>spec</td>
<td>Additional specificity for Rails 5.1</td>
<td>2 months ago</td>
</tr>
<tr>
<td>.gitignore</td>
<td>Codacy</td>
<td>11 months ago</td>
</tr>
<tr>
<td>.rspec</td>
<td>don't rebuild in the middle of prepending siblings.</td>
<td>3 years ago</td>
</tr>
<tr>
<td>.travis.yml</td>
<td>prep 6.5.0, add ar 5.1 appraisal</td>
<td>4 months ago</td>
</tr>
<tr>
<td>.yardopts</td>
<td>switch from rdoc to yard</td>
<td>6 years ago</td>
</tr>
<tr>
<td>Appraisals</td>
<td>prep 6.5.0, add ar 5.1 appraisal</td>
<td>4 months ago</td>
</tr>
<tr>
<td>CHANGELOG.md</td>
<td>prep v6.6.0</td>
<td>2 months ago</td>
</tr>
<tr>
<td>Gemfile</td>
<td>Bye bye rails 4.0</td>
<td>2 years ago</td>
</tr>
<tr>
<td>MIT-LICENSE</td>
<td>* version bump</td>
<td>a year ago</td>
</tr>
<tr>
<td>README.md</td>
<td>s/mceachen/ClosureTree/</td>
<td>2 months ago</td>
</tr>
</tbody>
</table>
class Employee < ActiveRecord::Base
  has_closure_tree
end
Person.find_or_create_by_path(
  [{name: 'Grandparent'},
   {name: 'Parent'},
   {name: 'Child'}
  ]
)
Mutation:

- create: 2 INSERTs
- reparent: 3 INSERT/UPDATEs
closure_tree (the gem)

- Speed
- No recursion
- Referential integrity
- Single table inheritance
- Deterministic ordering
- MySQL, PostgreSQL, and SQLite
- Easy migration
closure_tree

- No pgbouncer (concurrency is hard)
- Possibly not as fast as...
Materialized Path
<table>
<thead>
<tr>
<th>id</th>
<th>path</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/</td>
<td>/</td>
</tr>
<tr>
<td>2</td>
<td>1/2/</td>
<td>music</td>
</tr>
<tr>
<td>3</td>
<td>1/3/</td>
<td>photos</td>
</tr>
<tr>
<td>4</td>
<td>1/2/4/</td>
<td>rap</td>
</tr>
<tr>
<td>5</td>
<td>1/2/5/</td>
<td>rock</td>
</tr>
<tr>
<td>6</td>
<td>1/2/6/</td>
<td>polka</td>
</tr>
</tbody>
</table>
Descendents

```
SELECT *
FROM folders f
WHERE f.path LIKE '1/4/%';
```
Organise ActiveRecord model into a tree structure

<table>
<thead>
<tr>
<th>Branch</th>
<th>New pull request</th>
<th>Create new file</th>
<th>Upload files</th>
<th>Find file</th>
<th>Clone or download</th>
</tr>
</thead>
</table>

- **kbrock committed on GitHub** Merge pull request #357 from tilsammans/readme-generate-index-migration

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>gemfiles</td>
<td>Dropping official support for rails 4.0 and 4.1</td>
<td>4 months ago</td>
</tr>
<tr>
<td>lib</td>
<td>Fix order_by_ancestry_and</td>
<td>a month ago</td>
</tr>
<tr>
<td>test</td>
<td>mac specific tests</td>
<td>a month ago</td>
</tr>
<tr>
<td>.coveralls.yml</td>
<td>Fixing coveralls</td>
<td>4 years ago</td>
</tr>
<tr>
<td>.gitignore</td>
<td>Ignore locked gemfiles</td>
<td>2 years ago</td>
</tr>
<tr>
<td>.travis.yml</td>
<td>Add ruby 2.4 support in travis.</td>
<td>4 months ago</td>
</tr>
<tr>
<td>Appraisals</td>
<td>Dropping official support for rails 4.0 and 4.1</td>
<td>4 months ago</td>
</tr>
<tr>
<td>CHANGELOG.md</td>
<td>bump 3.0.1</td>
<td>2 months ago</td>
</tr>
<tr>
<td>Gemfile</td>
<td>ruby 1.9.3 support</td>
<td>11 months ago</td>
</tr>
<tr>
<td>MIT_LICENSE</td>
<td>Update copyright date</td>
<td>a year ago</td>
</tr>
</tbody>
</table>
ancestry

- Limits ~ 23 level deep
- No referential integrity
Ancestors x 10 runs
ancestry       0.01
closure_tree    0.03

Descendents x 10 runs
ancestry       0.014
closure_tree    0.02
But...

descendants.count

Closure tree is ~175X faster!
Focus on the reads!
Polymorphic Relationship
<table>
<thead>
<tr>
<th>id</th>
<th>holder_id</th>
<th>holder_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>SalesOrder</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>SalesOrder</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>SalesOrder</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>PurchaseOrder</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>PurchaseOrder</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>PurchaseOrder</td>
</tr>
</tbody>
</table>
Advantages

- Built into Rails
- Easy to add more
Disadvantages

- No foreign key
- No cascading delete
- Confusing column names
- 2 join conditions
- holder_type is extra data
Separate Tables
It's OK to not be DRY
Exclusive Arcs
<table>
<thead>
<tr>
<th>id</th>
<th>comment</th>
<th>post_id</th>
<th>photo_id</th>
<th>invitation_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>...</td>
<td>1</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
<td>2</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td>NULL</td>
<td>NULL</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>...</td>
<td>NULL</td>
<td>NULL</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
<td>NULL</td>
<td>NULL</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>...</td>
<td>NULL</td>
<td>4</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Advantages

- Foreign key constraints
- Cascading delete
Disadvantages

- Possible mistakes
- More migrations
Base Parent Table
1 to 1  1 to Many

photos
- id
- commentable_id
- filename

posts
- id
- commentable_id

invitation
- id
- commentable_id
- event_name

commentables
- id
- author_id
- posted_at
- message

comments
- id
- commentable_id
- message
Advantages

- No mistakes
- DRY
Disadvantages

- 2 Joins
- 1-way street
- Errors still possible
Dozens of possible parent types....
Dozens of possible parent types….

POLYMORPHIC RELATIONSHIP
Simple and *mostly* safe...
Simple and *mostly* safe...

EXCLUSIVE ARCS
Completely safe...
Completely safe...

BASE PARENT TABLE
Performance vs. Simplicity
Correctness vs. Convenience
MAKE GOOD CHOICES
I avoided recursive SQL Hell!
Resources

Who am I?

I tweet at @BradUrani

I work in Santa Barbara at linkedin.com/in/bradurani